TP1: Algoritmos de búsqueda en Torre de Hanoi

Autores: Mauro Barquinero, Yandri Uchuari, Marck Murillo, Matias Tripode

Enunciado

En clase presentamos el problema de la **torre de Hanoi**. Además, vimos diferentes algoritmos de búsqueda que nos permitieron resolver este problema. Para este trabajo práctico, deberán implementar un método de búsqueda para resolver con 5 discos, del estado inicial y objetivo.

Tareas y preguntas a resolver

- **¿Cuáles son los PEAS de este problema? (Performance, Environment, Actuators, Sensors)**
    - **Medida de Desempeño**: Resolver el problema en el menor número de movimientos posible.
    - **Entorno**: El entorno es un espacio de tres varillas, un número determinado de discos que varía de tamaño y el lenguaje python.
    - **Actuadores**: Los actuadores son la función de movimiento, discos y salidas del programa (ActionHanoi(), prints, JSON generado).
    - **Sensores**: El sensor lógico del agente le permite conocer el estado actual del entorno (es decir, la posición de cada disco en cada torre)..
- **¿Cuáles son las propiedades del entorno de trabajo?**
    - **Observabilidad**: El entorno es completamente observable. El agente puede ver toda la configuración de los discos en las tres torres en cualquier momento.
    - **Determinismo**: Es un entorno determinista, ya que cualquier acción que el agente realice tiene un resultado predecible.
    - **Episodicidad**: Es un entorno secuencial, no episódico. Cada acción afecta al estado del entorno de forma acumulativa, y las decisiones actuales dependen de la configuración previa de los discos en las torres.
    - **Dinamicidad**: El entorno es estático, ya que no cambia a menos que el agente realice una acción.
    - **Discretización**: El entorno es discreto porque el número de estados posibles (posiciones de los discos en las torres) es finito.
    - **Cantidad de agentes**: Es un entorno con un solo agente.
- **En el contexto de este problema, establezca cuáles son los: estado, espacio de estados, árbol de búsqueda, nodo de búsqueda, objetivo, acción y frontera.**

- **Estado**: Un estado es una configuración específica de los discos en las tres torres.
- **Espacio de Estados**: El espacio de estados es el conjunto de todas las configuraciones posibles de los discos en las tres torres.
- **Árbol de Búsqueda**: El árbol de búsqueda representa todas las secuencias posibles de movimientos de discos desde el estado inicial hasta el estado objetivo.
- **Nodo de Búsqueda**: Un nodo de búsqueda es un punto específico en el árbol de búsqueda que contiene un estado particular del problema.
- **Objetivo**: El estado objetivo es una configuración específica en la que todos los discos están apilados en la tercera torre.
- **Acción**: Una acción es un movimiento válido de un disco de una torre a otra. Una acción está restringida por las reglas.
- **Frontera**: La frontera es el conjunto de nodos de búsqueda que el agente ha generado pero que aún no ha explorado completamente.

Implementacion

Para el resolver el problema de las **torres de hanoi** se decidió implementar tres algoritmos de búsqueda, de las cuáles, dos ellos son no informadas(DFS y DFS Limitada) y uno informada (A*). La implementación de estos algoritmos se realiza con el objetivo de poder tener una comparación en la métricas de resolución del problema de las torres de hanoi

- ¿Qué complejidad en tiempo y memoria tiene el algoritmo elegido?

  Sea b=2 la cantidad de nodos en cada expansión y sea **d** la profundidad, entonces:

  DFS

  - Ejecución **O**(2^d)
  - Espacio **O**(d)

  DFS limitada

  - Ejecución **O**(k2^d), donde k es la iteración máxima
  - Espacio **O**(d)

  A*

  - Ejecución **O**(2^d),

- Espacio **O**(2^d). Debido a que mantiene los nodos en memoria (colas de prioridad de coste) para asegurar que no pierda el camino óptimo

A nivel implementación, ¿qué tiempo y memoria ocupa el algoritmo?).

DFS

- Tiempo promedio que demoró: **0.0139 [s]**
- Máxima memoria ocupada promedio: **0.13 [MB]**

DFS Limitada

- Tiempo promedio que demoró: **0.5571 [s]**
- Máxima memoria ocupada promedio: **0.17 [MB]**

A*

- Tiempo promedio que demoró: **0.968 [s]**
- Máxima memoria ocupada promedio: **0.36 [MB]**

- Si la solución óptima es 2^k - 1 movimientos con *k* igual al número de discos. Qué tan lejos está la solución del algoritmo implementado de esta solución óptima.

  En este caso la solución óptima es 2^5 - 1 = **31**

  DFS

  - El algoritmo ejecuta **81** movimientos lo cual es **2.6** veces más que la solución óptima.

  DFS Limitada

  - El algoritmo ejecuta **45** movimientos lo cual es **1.4** veces más que la solución óptima.

  A*

  - El algoritmo ejecuta **31** movimientos lo cual alcanzó la solución óptima.

Experimento

Número de ejecuciones: **30**

Algoritmo	# de caminos expandidos	**Tiempo [s]**	**Memoria [MB]**	**Longitud Camino**
DFS (con conjunto de nodos visitados)	**82**	**0.0139**	**0.13**	**81**
DFS Limitada (con conjunto de nodos visitados)	**65**	**0.5571**	**0.17**	**45**
A* (con conjunto de nodos visitados)	**177**	**0.0941**	**0.36**	**31**
BFS (con conjunto de nodos visitados) ← Provisto por la cátedra de IIA	**233**	**0.0365**	**0.23**	**31**

Comparación entre A*, BFS, DFS y DFS limitada

Caminos Expandidos:

- **A*** expandió **177 caminos**, mientras que **BFS** expandió **233 caminos**. Esto significa que A* fue más eficiente al explorar menos nodos para encontrar la solución óptima. Esto se debe a su heurística informada, que dirige la búsqueda hacia los nodos más prometedores, evitando expandir caminos que no contribuyen directamente a alcanzar la meta.
- **DFS** expandió **82 caminos** y **DFS limitada** expandió **65 caminos**, ambos menos que A* y BFS. Esto es característico de DFS, que explora un camino completo hasta el fondo antes de retroceder, lo cual resulta en menos expansiones en problemas profundos como este. Sin embargo, esto no garantiza que los caminos expandidos conduzcan a la solución óptima.

Tiempo de Ejecución:

- **A*** tuvo un tiempo promedio de **0.0941 segundos**, por lo que lo hace considerablemente más competitivo en términos de eficiencia temporal. Este tiempo es muy cercano al **BFS**, lo cual hace a A* una opción favorable dado su menor número de expansiones
- **BFS** tomó **0.0365 segundos**, siendo más rápido que A* debido a su enfoque de exploración exhaustiva sin cálculos heurísticos.
- **DFS** fue el más rápido de todos, con un tiempo de **0.0139 segundos**. Esto se debe a que DFS explora un camino a fondo, lo que puede resultar en encontrar una solución rápidamente sin explorar muchos nodos, aunque no garantiza una solución óptima.
- **DFS limitada** tuvo un tiempo promedio de **0.5571 segundos**. Esto se debe a la profundidad limitada, que obliga al algoritmo a re-explorar caminos con mayor frecuencia al alcanzar el límite de profundidad, lo que incrementa el tiempo de ejecución.

Uso de Memoria:

- **A*** utilizó **0.36 MB** de memoria, en comparación con **0.23 MB** para **BFS**. Aunque A* usa más memoria, esta se justifica por el almacenamiento de costos acumulados y heurísticos que guían su búsqueda de manera informada.
- **DFS** usó **0.13 MB**, y **DFS limitada** usó **0.17 MB**, ambos menos que A* y BFS. Esto se debe a que DFS y DFS limitada no almacenan información detallada sobre cada nodo expandido. DFS solo necesita recordar el camino actual y el límite de profundidad, por lo que utiliza menos memoria.

Longitud del Camino:

- **A*** y **BFS** alcanzaron **la solución óptima de 31 movimientos** (el mínimo teórico de $2^5 - 1 = 32 - 1 = 31$ para 5 discos). Esto muestra que ambos algoritmos pueden encontrar la solución óptima, pero A* lo hace de manera más eficiente en términos de expansiones y tiempo.
- **DFS** encontró una solución de **81 movimientos**, lo cual es mucho mayor que la óptima, ya que DFS no garantiza encontrar el camino más corto.
- **DFS limitada** encontró una solución de **45 movimientos**, que es mejor que la de DFS completo, pero aún no es óptima. DFS limitada reduce la exploración infinita, pero al imponer un límite de profundidad, puede encontrar soluciones subóptimas.

Conclusiones

Cada algoritmo tiene sus ventajas y desventajas en función de los objetivos y las restricciones del problema:

- **DFS** es rápido y utiliza poca memoria, pero no garantiza una solución óptima y puede explorar caminos innecesariamente largos.
- **DFS Limitada** es útil cuando se necesita controlar la profundidad de búsqueda, pero sigue siendo subóptimo en términos de la solución encontrada.
- **A*** proporciona un equilibrio entre precisión y eficiencia temporal, alcanzando la solución óptima con menos expansiones de caminos y un tiempo de ejecución razonable de **0.0941 segundos**, aunque a costa de un mayor uso de memoria.

- **BFS** garantiza la solución óptima y utiliza menos memoria que A*, pero es menos eficiente en términos de caminos expandidos y tiempo de ejecución debido a su exploración exhaustiva.

Realizado las comparaciones en las métricas de los algoritmos implementados. podemos concluir que los algoritmos **A*** y **BFS** fueron los únicos que alcanzaron la solución óptima de 31 movimientos. **A*** se destaca como una opción equilibrada, mostrando eficiencia tanto en la cantidad de caminos expandidos como en su tiempo de ejecución, aunque requiere más memoria para mantener su estrategia informada y dirigida.